REMARKS

I. Introduction

In response to the Office Action dated July 24, 2007, which was made final, and in conjunction with the Request for Continued Examination (RCE) submitted herewith, claims 1, 7 and 13 have been amended. Claims 1-18 remain in the application. Re-examination and re-consideration of the application, as amended, is requested.

II. Non-Art Objection:

On page 2 of the Office Action, claim 10 was objected to as having an improper status identifier.

Applicants' attorney has included the proper status identifier for claim 10.

III. Prior Art Rejections

A. The Office Action Rejections

On pages 2-9 of the Office Action, claims 1-18 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,948,113 (Johnson) in view of U.S. Patent No. 6,785,848 (Glerum).

Applicants' attorney respectfully traverses these rejections.

B. The Applicants' Independent Claims

Independent claims 1, 7 and 13 are generally directed to providing contextual diagnostic data at a point of failure of a software program. Independent claims 1, 7 and 13 have all been amended to better distinguish over the references.

Claim 1 is representative and recites a method for providing contextual diagnostic data at a point of failure of a software program, comprising:

(a) registering one or more callback functions for each of one or more modules and sub-applications within the program;

(b) examining a call stack for the program upon failure of the program;

(c) notifying the registered callback functions for the modules and sub-applications based on the examined call stack, wherein an error handler gives each module or sub-application that registered a callback function and is on the examined call stack an opportunity to include specific

diagnostic information based on the examined call stack and based on the point of failure within the module or sub-application;

(d) performing callback processing, wherein the notified callback functions of the modules and sub-applications interpret the call stack's context supplied to the callback function, determine the contextual diagnostic data to be extracted and supplied to the error handler, and extract and supply the contextual diagnostic data;

(e) packaging the contextual diagnostic data supplied by the notified callback functions of the modules and sub-applications; and

(f) using the packaged contextual diagnostic data for further analysis in order to troubleshoot the point of failure of the software program.

C. The Johnson Reference

Johnson describes centrally handling a runtime error or exception of a program using a central object stack and exception handling code centrally maintained within a global object manager. The global object manager is a data structure separate from the program's call stack. When a modified TRY statement is executed, a location is marked on the central object stack. During execution of a section of code after the modified TRY statement, if a new object is needed, the global object manager efficiently allocates the new object. The global object manager either allocates the new object directly from memory or attempts to re-use a previously allocated object in a cache of available objects as the new object. The new object is then registered on the central object stack and a pointer to the new object is registered on the program's call stack. This keeps the new object and associated exception handling code off the program's call stack. When an exception is thrown, the global object manager cleans up and unregisters an object which was registered on the central object stack since the marked location. If a re-use condition is met, the object is kept in the cache as an available object already allocated from memory. However, if the re-use condition is not met, the object is de-allocated from memory.

D. The Glerum Reference

Glerum describes a method for categorizing information regarding a failure in an application program module. The failure may be a crash, a set-up failure or an assert. For a crash, a name of an executable module where the crash occurred in the application program module, a version number of the executable module, a name of a module containing an instruction causing the crash, a version

number of the module and an offset into the module with the crashing instruction are determined. This bucket information is then transmitted to a repository for storage in a database. The database may be examined to determine fixes for the bug that caused the crash.

E. The Applicants' Invention is Patentable Over the References

The Applicants' invention, as recited in amended independent claims 1, 7 and 13, is patentable over the combination of Johnson and Glerum references, because it contains limitations not taught by the combination of references.

For example, with regard to the limitations "registering one or more callback functions for each of one or more modules and sub-applications within the program," "examining a call stack for the program upon failure of the program," and "notifying the registered callback functions for the modules and sub-applications based on the examined call stack, wherein an error handler gives each module or sub-application that registered a callback function and is on the examined call stack an opportunity to include specific diagnostic information based on the examined call stack and based on the point of failure within the module or sub-application," the cited locations in Johnson are set forth below:

> Johnson: col. 9, lines 26-45
>
> To implement exception handling, the GOM 200 maintains a central object stack 220 of registered objects currently in use, preferably on a per-thread basis. The GOM 200 places or registers allocated objects on the thread's central object stack 220. Prior to entering the section of code where an exception may occur, the current location (stack address) on the central object stack 220 is marked and stored. If an exception occurs within the section of code, the GOM 200 handles the exception using the central object stack 220 and the centrally maintained exception handling code 210.
>
> Essentially, the exception handling code 210 within the GOM 200 pops off objects 225a-c on the central object stack 220 added or registered since the stored location or address. In this manner, the objects popped off the central object stack 220 are deemed to be unregistered from the central object stack 220. The memory (e.g., the system memory 22) associated with each of the objects are cleaned up and the object is either saved for re-use or de-allocated from system memory 22 before continuing with execution of the section of code.

The above portions of Johnson refer to a central object stack of registered objects, but the central object stack is not a call stack of the program. Indeed, the Johnson reference notes that, traditionally, objects are allocated on an application's call stack as they are needed when the program code of the application is executed, but that the central object stack is independent and separate

from the application's call stack. The central object stack of Johnson is merely used to perform clean-up of objects in use when an exception occurs.

However, Johnson says nothing about registering callbacks for each of the modules and sub-applications within the program, examining a call stack for the program upon failure of the program and then notifying the registered callback functions for the modules and sub-applications based on the examined call stack, wherein an error handler gives each module or sub-application that registered a callback function and is on the examined call stack an opportunity to include specific diagnostic information based on the examined call stack and based on the point of failure within the module or sub-application.

Instead, the Johnson reference is merely concerned with exception related cleanup of allocated objects in traditional try-catch blocks, using a central object stack separate from the call stack.

The Office Action, on the other hand, asserts the following:

> Applicant further asserts on page 8 of the amendment that the central object stack in Johnson is not a call stack of the program.
>
> Examiner respectfully disagrees with the allegation as argued. Although, the central object stack is independent and separate from the application's call stack as disclosed by Johnson, but the use of the central object stack is the same as the call stack as disclosed in the instant application. To reduce the burden of dealing with the program's call stack, Johnson's approach using the central object stack. However, when the object registers on the central object stack, a pointer to the object is also registered on the program's call stack (see at least the abstract). Another words, the object is also registered on the program's call stack, but the central object stack is used for handling a runtime error or exception of a program to reduce the burden of dealing with the program's call stack. Even though, the central object stack is independent and separate from the application's call stack, they are the same.

In essence, the Office Action admits that the central object stack in Johnson is not a call stack of the program, but then contradicts itself and asserts that the central object stack and the call stack are the same.

However, the central object stack in Johnson merely provides a mechanism for "cleaning up" objects added or registered on the central object stack should an exception occur and is merely an adjunct to a call stack, but is not a call stack and thus does not read on Applicants' claim limitations directed to a call stack.

Indeed, there is nothing in Johnson describing the central object stack being used to notify the registered callback functions for the modules and sub-applications based on the examined call

stack, wherein an error handler gives each module or sub-application that registered a callback function and is on the examined call stack an opportunity to include specific diagnostic information based on the examined call stack and based on the point of failure within the module or sub-application.

In another example, with regard to the limitations "performing callback processing, wherein the notified callback functions of the modules and sub-applications interpret the call stack's context supplied to the callback function, determine the contextual diagnostic data to be extracted and supplied to the error handler, and extract and supply the contextual diagnostic data," and "packaging the contextual diagnostic data supplied by the notified callback functions of the modules and sub-applications," the cited locations in Glerum are set forth below:

> Glerum: col. 5, lines 47-67
>
> The system 200 also comprises an exception filter 220. Exception filters are well-known in the art and may be registered by program modules when the operating system 35 is started. When a failure (an exception) occurs, the exception filter 220 code is executed. For example, suppose a failure occurs while executable program 210 is executing instructions running module 215 at location 225. If executable program 210 has registered exception filter 220 with the operating system, then the exception filter 220 is executed when executable program 210 encounters an exception.
>
> In the system 200, exception filter 220 executes a failure reporting executable 230. The failure reporting executable 230 is an executable program comprising all of the instructions needed to communicate between the application program module 205 and a repository 235. The communications between the failure reporting executable 230, the application program module 205 and the repository 235 are illustrated as arrows in FIG. 2. The failure reporting executable 230 is preferably separate from the application program module 205 because of the possible instability of the application program module (having experienced a failure).

> Glerum: col. 6, lines 32-47
>
> Based upon the type of failure, the failure reporting executable 230 then determines what relevant information to retrieve from the application program module to uniquely identify, i.e. categorize, the failure. In many cases, uniquely identifying the failure means determining the location of the failure. Typically, the categorization, or location information, of the failure is sent to the repository as a bucket. A bucket is a set of information uniquely defining the location of the failure. If a bucket from one failure matches a bucket from another failure, then it is assumed that both failures are caused by the same bug. Although not always accurate (because more than one bug may be at the same location), this assumption that failures with the same bucket information are caused by the same bug allows for effective organization in the repository, as will be further described below.

The above portions of Glerum merely describe an exception filter, which is for the type of exception being handled. The exception filter then executes a failure reporting executable that, based upon the type of failure, determines what relevant information to retrieve from the application program module to identify or categorize the failure.

However, Glerum says nothing about performing callback processing, wherein the notified callback functions of the modules and sub-applications interpret the call stack's context supplied to the callback function, determine the contextual diagnostic data to be extracted and supplied to the error handler, and extract and supply the contextual diagnostic data, in the context where the callback functions are registered for each of the modules and sub-applications within the program, the call stack for the program is examined upon failure of the program and the registered callback functions for the modules and sub-applications are notified based on the examined call stack, wherein an error handler gives each module or sub-application that registered a callback function and is on the examined call stack an opportunity to include specific diagnostic information based on the examined call stack and based on the point of failure within the module or sub-application.

Instead, the Glerum reference merely describes typical exception handling by exception type, along with minidumps, bucketing and typical error reporting.

The Office Action, on the other hand, asserts the following:

> Applicant asserts on page 10 or the amendment that Glerum says nothing about performing callback processing, wherein the notified callbacks of the modules and subapplications extract and supply the contextual diagnostic data.
>
> Examiner respectfully disagrees with this allegation as argued. Glerum discloses "if the executable program 210 has registered exception filter 220 with the operating system, then the exception filter 220 is executed (called) when the executable program 210 encountered an exception" (see at least col. 5, lines 53-56).
>
> Glerum further discloses "the exception 220 executes the failure reporting executable 230. The failure reporting executable 230 determines what relevant information to retrieve from the application program module to uniquely identify, i.e. categorize, the failure" (see at least col. 6, lines 23-47). Another words, failure data is gathering for supplying to the repository.

In making this argument, the Office Action misconstrues Applicants' claims when comparing them to Glerum.

In Glerum, only a single exception filter is executed when an error occurs, wherein the exception filter executes a failure reporting executable. However, there are no callback functions registered for each of the modules and sub-applications in the program. Moreover, the failure

reporting executable determines what relevant information to retrieve based upon the type of failure, not based on the point of failure, as recited in Applicant's claims.

There is nothing in Glerum describing the central object stack being used to notify the registered callback functions for each of the modules and sub-applications based on the examined call stack, wherein an error handler gives each module or sub-application that registered a callback function and is on the examined call stack an opportunity to include specific diagnostic information based on the examined call stack and based on the point of failure within the module or sub-application

Thus, the combination of Johnson and Glerum does not render obvious Applicants' claimed invention. Moreover, the various elements of Applicants' claimed invention together provide operational advantages over the combination of Johnson and Glerum. In addition, Applicants' claimed invention solves problems not recognized by the combination of Johnson and Glerum.

Thus, Applicants' attorney submits that independent claims 1, 7 and 13 are allowable over the combination of Johnson and Glerum. Further, dependent claims 2-6, 8-12 and 14-18 are submitted to be allowable over the combination of Johnson and Glerum in the same manner, because they are dependent on independent claims 1, 7 and 13, respectively and thus contain all the limitations of the independent claims. In addition, dependent claims 2-6, 8-12 and 14-18 recite additional novel elements not shown by the combination of Johnson and Glerum.

IV. Conclusion

In view of the above, it is submitted that this application is now in good order for allowance and such allowance is respectfully solicited.

Should the Examiner believe minor matters still remain that can be resolved in a telephone interview, the Examiner is urged to call Applicants' undersigned attorney.

Respectfully submitted,

GATES & COOPER LLP
Attorneys for Applicants

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

Date: October 23, 2007

By: _____
Name: George H. Gates
Reg. No.: 33,500

GHG/

G&C 30566.315-US-01